DOCKET NO: 149-0105US
(02-014-US)

APPLICATION

FOR

UNITED STATES LETTERS PATENT

TITLE: Sorting Objects Having Variable Length Keys

INVENTOR: Christopher Y. Blaicher

Express Mail No: EV195559824US

Date: July 11, 2003

Prepared by: WONG, CABELLO, LUTSCH, RUTHERFORD & BRUCCULERI, L.L.P.
HOUSTON, TEXAS
(VOICE) 832-446-2400 (FACSIMILE) 832-446-2424

SORTING OBJECTS HAVING VARIABLE LENGTH KEYS

Background

[0001] The present invention relates to sorting techniques and more particularly, but not by way of limitation, to a system and methods for sorting data having variable length sort key fields.

[0002] The process of sorting an object generally involves (1) reading the object from external storage, often referred to as "unloading" the object, (2) passing the object's data to a utility that reorders/sorts the data in accordance with a specified sort key and (3) writing the sorted data back to the object's file on external storage, often referred to "reloading" the object. It will further be understood that all sort routines compare items of the same size. That is, the size of an object's sort key must be constant from record to record during the sort operation. As used herein, the term 'object' refers to any collection of data that may be sorted. For example, an array or list of elements, one or more tables in a relational database or a collection of records within a conventional file structure. One of ordinary skill in the art will understand that an object typically includes one or more records, that records are comprised of one or more fields, that one or more fields are designated as a sort key and that sorting reorders an object's records based on the value of the records' sort keys.

[0003] One prior art technique for sorting objects having variable length keys is shown in FIG 1. Sort routine **100** reads and pads a record from the object

being sorted (block **105**). The act of padding converts variable length key fields to fixed length key fields of a size great enough to accommodate any value that the key may assume. Once padded, the record is written to an intermediate file (block **110**). If there are additional records to pad (the 'NO' prong of block **115**), processing continues at block **105**. If the object has been completely unloaded (the 'YES' prong of block **115**), a sort utility is invoked that reorders and then stores the padded records in a result file (block **120**). Each sorted and padded record is then retrieved from the result file, unpadded and reloaded into the object (blocks **125**, **130** and **135**). The act of unpadding adjusts the size of each record's sort key to its original size. Using pad and unpad processes can be very intensive to intermediate storage since a padded copy of the entire object (possibly an entire database) needs to be created. Given that many keys are a fraction of the size that must be supported; the padded copy of an object can be several times the size of the original. In addition, since a single object can not generally be retained in working memory, the time required to write and read an intermediate file having expanded sort keys can consume a significant portion of the total time needed to sort the object (e.g., the total elapsed time from block **105** to block **135**).

[0004] To mitigate some of the aforementioned drawbacks, certain commercially available sort applications support input and output routine customization. These user-created, application specific, programs are commonly referred to as E15 (input) and E35 (output) programs. As illustrated in FIG. 2,

after obtaining a record from (unsorted) object **200**, E15 program **205** pads the record's sort key and passes the padded record to E15 interface **210**. Sort routine **215** accepts input from E15 interface **210**, sorts the records in accordance with the specified key and manages the transfer of padded data to and from intermediate storage **220**. Following completion of the sort operation, sort routine **215** passes the sorted and padded records through E35 interface **225** to E35 program **230** which then unpads the records and reloads object **200**.

[0005] One significant drawback to prior art sorting techniques is that for large objects (e.g., databases and/or large database objects) comprising tens of megabytes to tens of terabytes, the time required to transfer padded data to and from intermediate storage can comprise a significant portion of the total time required to unload, sort and reload the target object. In addition, the amount of intermediate storage needed to retain padded data can be a significant use of resources. Thus, it would be beneficial to provide techniques (methods and devices) to sort data that is more time and resource efficient than current techniques.

Summary

[0006] In one embodiment, the invention provides a method to sort data. The method includes obtaining a plurality of data records and, for each data record: (1) extracting key information, (2) expanding the extracted key information, and

(3) storing the expanded key information in a key record. The plurality of key records may then be sorted (based on the expanded key information), and used to reorganize the plurality of data records which may then be stored in working or intermediate storage while additional plurality of data records are similarly processed. Unlike prior art sorting techniques, methods in accordance with the invention do not store expanded key information to intermediate storage. Methods in accordance with the invention may be embodied in instructions stored in any media that is readable and executable by a computer system.

[0007] In another embodiment, the invention provides a sort control card that may be used to identify the location of key fields within a data record. Sort control cards in accordance with the invention may be particularly useful in those situations in which data records comprise two or more variable length fields (key or data fields). Sort control cards in accordance with the invention may also be particularly useful in those situations in which fixed length key fields are used, but where one or more variable length data fields precede one or more key fields.

Brief Description of the Drawings

[0008] Figure 1 shows one prior art method for sorting an object.

[0009] Figure 2 shows another prior art method for sorting an object.

[0010] Figure 3 shows, in flowchart form, a sorting technique in accordance with one embodiment of the invention.

[0011] Figure 4 shows, in flowchart form, a key extraction technique in accordance with one embodiment of the invention.

[0012] Figure 5 shows an illustrative sort control card in accordance with one embodiment of the invention.

[0013] Figure 6 shows, in block diagram form, a merge operation in accordance with one embodiment of the invention.

[0014] Figure 7 shows, in block diagram form, a preferred embodiment of the invention.

Detailed Description

[0015] The invention relates generally to sorting and, more particularly, to methods and devices for sorting data having variable length keys. Compared to the prior art, techniques in accordance with the invention can substantially reduce the amount of data moved between working memory and intermediate storage during a sort operation. This, in turn, can significantly reduce the start-to-finish time and the amount of intermediate storage needed by the sorting operation. As previously noted, as used herein the term 'object' refers to any collection of data that may be sorted and includes, without limitation, database objects, arrays, lists and collections of records within conventional file structures.

[0016] Figure 3 illustrates sorting process **300** in accordance with one embodiment of the invention. To begin, a first quantum of object data is retrieved, generally comprising a number of records (block **305**). A quantum of

data is typically that amount of data that can be retained and manipulated in working memory such as, for example, in a data-in buffer. Key extraction techniques are then used to identify the fields within a record that comprise the object's sort key and the maximum size of each of the fields; the relevant keys from each record obtained are then extracted, padded and placed into a key structure (block **310**). Key structures in accordance with the invention generally include one record for each data record retrieved in accordance with block **305**, wherein each key structure record further includes an indication of the data record from which the key was extracted. The key structure is then sorted to reorder the key structure records in accordance with the desired sort properties (block **315**). For example, the key structure records may be reordered in ascending order based on the value of the padded key values. The sorted key structure is then used to reorder the data obtained during the acts of block **305** (block **320**).

[0017] If the object being sorted has data that has not yet been processed (the 'YES' prong of block **325**), the reorganized quantum of data is written to intermediate storage (block **330**) and the next quantum of data is retrieved (block **305**). The acts of block **310-320** are repeated for each quantum of object data. It is significant to note that the reorganized data written to intermediate storage in accordance with block **330** does not include padded key information. Accordingly, both the amount of data and the time required to store and retrieve this data is substantially less than that of prior art sorting

techniques. If all of the object's data has been processed in accordance with blocks **305- 320** (the 'NO' prong of block **325**), the one or more quanta of sorted data are reloaded back into the database (block **335**), thereby completing the object's reorganization. It will be recognized that if the data object being sorted comprises more than one quanta of data, the intermediate sorted quanta (written during the acts of block **330**) must first be merged prior to, or in the process of, reloading the object in block **335**.

[0018] In one embodiment, the acts of block **310** (see FIG. 3) may be embodied as illustrated in FIG. 4. To begin, records from a data object obtained in accordance with block **305** of FIG. 3 are retrieved into a temporary working memory such as, for example, data-in buffer **400**. As shown, data object records **405** and **410** may be different lengths. For each record retrieved from buffer **400** (block **415**), a first field is identified as a key field (block **420**). The identified field is then copied into a working memory and padded to its maximum length (block **425**). If the record has additional key fields (the "YES" prong of block **430**), the acts of blocks **420** and **425** are repeated. If the record has no additional key fields (the "NO" prong of block **430**), the padded key fields are saved to key structure **440** (block **435**). If the data object has additional records (the "YES" prong of block **445**), the acts of block **415-435** are repeated. If the data object has no additional records (the "NO" prong of block **445**), key extraction in accordance with the invention (e.g., block **300**) is complete.

[0019] As shown, each data object record in buffer **400** (e.g., record-1 **405** and record-N **410**) has a corresponding record in key structure **440** (e.g., key record-1 **445** and key record-N **450**), wherein each key record comprises a padded key field (e.g., **460** and **465**), a data pointer field (e.g., **470** and **475**) and a next pointer field (e.g., **480** and **485**). Padded key fields **460** and **465** represent the combined padded key fields for each record and, as such, are of fixed length. Data pointer fields **470** and **475** point back to that data record from which the key record's padded key field was extracted. That is, data pointer field **470** points to record-1 **405** and data pointer field **475** points to record-N **410**. Next pointer fields **480** and **485** point to the next record in key structure **440**.

[0020] Referring now to FIGS. 3 and 4, a key structure populated in accordance with FIG. 4, may be sorted in accordance with block **315** (see FIG. 3). Once sorted, the key structure's data pointer fields may be used to reorganize the data records in buffer **400** in accordance with block **320** (see FIG. 3). As the next buffer of data object records are loaded into working memory (i.e., buffer **400**), the newly reorganized buffer of records may be written to working or intermediate storage (see block **330** of FIG. 3). In another embodiment, the data records may be reorganized into a second (e.g., output) buffer which may then be sent to working or intermediate storage. It is significant to note that the data written to intermediate storage does not include padded key field information. Accordingly, the amount or volume of data written

to working storage and the time required to write (and subsequently read) it can be significantly less than in prior art techniques, especially in the aggregate for large data objects.

[0021] Key identification and extraction in accordance with one embodiment of the invention uses sort control cards that identify where each key is within a data record (e.g., record-1 **405** of FIG. 4), the maximum size of the key and the field's data type.. (It is noted that prior art technology does not permit the creation of key structures from object data having variable length keys or keys that start at arbitrary locations within a record.) One illustrative example of a sort control card that characterizes a data object's fields in a manner that enables the identification of the object's sort fields is as follows:

[0022] SORT_FIELDS(S1, M1, T1, A1 ... Sn, Mn, Tn, An).

[0023] S1 represents the starting location in the record of the first key field and Sn represents the starting location of the nth field. If the nth field's starting location varies depending upon the actual length of prior fields (e.g., the first key field), Sn may be set to a value indicative of this such as, for example, an asterisk ('*'). M1 represents the maximum length of the first sort field and Mn represents the maximum length of the nth field. (One of ordinary skill in the art will recognize that the actual size of any given field in a record may be encoded in the field itself or in another known location within the record.) T1 represents the type and nature of the first sort field and Tn the type and nature of the nth field. Illustrative field types include, but are not limited to, floating point, integer,

character, Boolean and ASCII types. In addition, the type of field may be augmented to indicate whether it is a fixed-length or variable-length field. For example, a fixed-length character field may be identified as FCH while a variable length character field may be identified as VCH. A1 and An indicate whether the first and nth fields (respectively) should be sorted in an ascending or descending order. As illustrated below, if a field is not a key field it may not have an associated sequence (ascending/descending) value. Using a sort control card such as that described above, the object to be sorted can be queried to obtain the object's key information so that the object's keys may be properly extracted and padded.

[0024] Referring to FIG. 5, illustrative sort control card **500** identifies five (5) fields in a data object's record. The first field, key-1 **505**, begins at a fixed position (27) in the record. (This implies there are no variable length fields before position 27.) The maximum length of field **505** is 35 bytes, it is a variable length character type (VCH) and should be sorted in ascending order (A). The second field, data-1 **510**, begins at a variable position within the record ('*'), is 10 bytes in length and is data (DATA) – that is, not a key. Because field **510** is not a key field, it does not require a sort-order indication attribute. It will be recognized by one of ordinary skill in the art that the precise starting location of data-1 field **510** may be calculated based on the known starting position of prior field **505** and the known actual size of filed **505**, which is typically encoded as the first one or more bytes in field **505**. (This same process may also be used to

calculate the actual starting location for each subsequent filed – **515**, **520** and **525**.) The third field, key-2 **515**, is another key field that begins at a variable position within the record ('*'), has a maximum length of 8 bytes, is a variable length long integer (VL) and should be sorted in ascending order (A). The fourth field, data-2 **520**, begins at a variable position within the record ('*'), has a maximum length of **255** bytes and is a variable length non-key field (VDATA). The fifth field, key-3 **525**, is yet another key field that begins at a variable position within the record ('*'), has a maximum length of 55 bytes, is a variable length binary field (VBI) and should be sorted in descending order (D).

[0025] It will be recognized that most practical data objects are so large that they cannot be sorted in a single pass. That is, a single data object may comprise sufficient information to fill many hundreds, thousands or millions of buffers (e.g., buffer **400** of FIG. 4). In these cases, those portions of the data object sorted and written to intermediate storage during the acts of block **330** of FIG. 3, must be merged before the object may be reloaded in accordance with block **335** of FIG. 3.

[0026] Referring now to FIG. 6, in one embodiment sorted data object information is retrieved from intermediate storage **600** and placed into two or more input buffers (e.g., **605** through **610**). Because each input buffer of data **605** through **610** has already been sorted, its lowest (highest) key value is known. For example, the record having the lowest (highest) key value may be the first record in each buffer. To permit comparison, the designated key value

(lowest or highest) is extracted (see, for example, the discussion above regarding FIGS. 3 and 4), padded and placed in a key register (e.g., **615** through **620**). The keys are then evaluated by compare operator **625**, with the record having the lowest (highest) key value being placed in output buffer **630**. This operation is repeated until all records in buffer **605** through **610** are merged. One of ordinary skill in the art will recognize that the operations of FIG. 6 may be repeated a number of times and that, further, such operations may involve using intermediate storage. That is: a first plurality of data buffers may be merged into a first output buffer which is then written to intermediate storage; a second plurality of data buffers may then be merged into a second output buffer which is also written to intermediate storage ... wherein the two or more intermediate output buffers are then merged in accordance with FIG. 6. Regardless, however, of the number of levels of comparison needed, the operations will proceed as outlined. In another embodiment, each key is expanded as it is brought in from intermediate storage **600** and placed in an input buffer. In this embodiment, the expanded key values are reduced to their original state prior to reloading object **635**.

[0027] A preferred embodiment in terms of a DB2® database can be seen in Figure 7. (DB2 is a registered trademark of the International Business Machines Corporation of Armonk, New York.) Data is retrieved from object **635** one record at a time by E15 program **700**. Each record is passed to E15 interface **705** which places them into buffer **710**. E15 Interface **705** extracts and expands

each record's key fields, storing the padded keys as fixed length components in a key record stored in key structure **715**. When input buffer **710** and/or key structure **715** is full (or until all of object **635**'s records are obtained), key structure **715** is sorted by sort routine **720**. Sorted key structure **715** is used to reorganize the records in buffer **710** (or, alternatively, into a separate output buffer) which is then stored on intermediate storage device(s) **600**. After all of object **635**'s records have been processed, control is passed to E35 interface **725** which merges the sorted data stored on intermediate storage **600** (block **730**). The merged data is passed to E25 program **735** which then reloads the data into object **635**.

[0028] In another embodiment, key extraction, pad and sort techniques in accordance with the invention may be implemented within sort routine **720** itself. In this latter embodiment, E15 program **700** and/or E15 interface **705** may simply provide, without modification, data object records to sort routine **720**. In still another embodiment, E15/E35 programs and interfaces may not be used at all. In these embodiments, sort routine **720** obtains data object records directly from object **635**.

[0029] While the invention has been disclosed with respect to a limited number of embodiments, numerous modifications and variations will be appreciated by those skilled in the art. For example, various changes in the details of the illustrated operational methods are possible without departing from the scope of the claims. By way of example, various key extraction and/or sort

control card techniques may be employed to identify and expand key fields within a data record. In addition, one of ordinary skill in the art will recognize that data pointer fields **470** and **475** and next pointer fields **480** and **485** may be embodied in non-pointer fields. For example, key structure **440** may be implemented as an array such that next pointer fields **480** and **485** may be eliminated. Similarly, if data-in buffer **400** is implemented as an array, data pointer fields **470** and **475** in key structure **440** may retain the index value (typically an integer) of the associated data-in array index. It will further be recognized that data objects (e.g., **635**) may be stored on one or more direct access storage devices and, also, that intermediate storage (e.g., **600**) may comprise one or more such devices. It is further noted that sorting techniques in accordance with the invention are not limited to sorting database objects, but instead may be applied to sort any data and is especially helpful when processing/sorting large data objects. One of ordinary skill in the art will also recognize that the sorting and key extraction techniques described herein are equally applicable to data having variable length keys and/or to data having one or more variable length data fields located prior to a (fixed or variable length) key field. That is, in both instances prior art techniques do not provide a way to locate or identify key field starting locations and, further, do not limit the amount of intermediate storage used by storing only raw (unpadded) object data.

[0030] Acts in accordance with FIGS. 3 and 4 and the operations associated with FIG. 6 may be performed by a programmable control device executing

instructions organized into a program module. A programmable control device may be a single computer processor, a plurality of computer processors coupled by a communications link, or a custom designed state machine embodied in a hardware device such as a printed circuit board comprising discrete logic, integrated circuits, or specially designed application specific integrated circuits (ASICs). Storage devices suitable for tangibly embodying program instructions include, but are not limited to: magnetic disks (fixed, floppy, and removable) and tape; optical media such as CD-ROM disks; and semiconductor memory devices such as Electrically Programmable Read-Only Memory (EPROM), Electrically Erasable Programmable Read-Only Memory (EEPROM), Programmable Gate Arrays and flash devices.